НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Фізико-математичний факультет
(повна назва інституту/факультету)

Кафедра загальної фізики та фізики твердого тіла
(повна назва кафедри)

«На правах рукопису»
УДК 538.911:004](043.3)

«До захисту допущено»

Завідувач кафедри
_____ В.Й. Котовський
(підпис) (ініціали, прізвище)

“___”_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації) 104 фізика та астрономія _____
(код і назва спеціальності)

_____

на тему: Нестійкості росту наноструктур із алмазоподібною граткою в дифузійному режимі _____

Виконав (-ла): студент (-ка) 2 курсу магістерського рівня, групи ОФ-71мп_
(шифр групи)

Стадніченко Григорій Олександрович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник д. фіз. мат. н., проф. Горшков В. М._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент д. фіз. мат. н., проф. Решетняк С. О. _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018

АНОТАЦІЯ

ДО ЗВІТУ З НАУКОВО-ДОСЛІДНОЇ ПРАКТИКИ

СТУДЕНТА ФМФ, 2 КУРСУ МАГІСТЕРСЬКОГО РІВНЯ ГР. ОФ-71мн

Стадніченка Григорія Олександровича

На тему: «Нестійкості росту наноструктур із алмазоподібною граткою в дифузійному режимі» **Науковий керівник** д. фіз. мат. н., проф. Горшков В. М.

Процеси дифузії у твердих тілах є предметом досліджень у галузі фізики твердого тіла. Дифузія відіграє визначальну роль у процесах росту тонких плівок, формуванні наноструктур на поверхні підкладок і спікання кераміки, тому має бути досліджена.

Оскільки окрім енергетичної складової процесу є іще кінетична, необхідно розглядати їх комбінацію в експериментах із формування нанокристалів. Не зважаючи на умови стійкості з точки зору енергії, деякі конфігурації можуть виявитись не пояснюваними за допомогою такого підходу. Так, наприклад, будь які нестійкості в кристалах є енергетично невигідними, але, тим не менш, виникають під час реальних експериментів.

У цій магістерський дисертації буде розглянуто особливості утворення нестійкостей на поверхнях матеріалів із алмазоподібною граткою на прикладі фізичної моделі на сонові методу Монте-Карло. Основною метою є дослідження умов розвитку нестійкостей за різних умов, а саме, на основних кристалографічних площинах, а також дослідження одночасної еволюції декількох кристалографічних площин. Виявлення можливих закономірностей між цими факторами грає визначну роль у розумінні процесів еволюції нанокристалів.Результатом проведеної роботи слугує комп’ютерна програма що поверхневі явища в наномасштабах. Отримано умови утворення нестійкостей.

Було проведено більше 100 експериментів з загальним обсягом даних порядку 200 гігабайт. В звіті наведено 20 рисунків, сторінок -83, літератури - 43.

Ключові слова: Дифузія, алазоподібна гратка, чисельна модель.

ANNOTATION

OF THE SCIENTIFIC AND RESEARCH PRACTICE

STUDENT FMF, 2 COURCE MASTERY GR. OF-71mn

Stadnichenko Hryhorii

Theme: «Instability of the growth of nanostructures with diamond-like lattice in diffusion mod»

Scientific supervisor: d. phys. math. sci., prof. Gorshkof V. M.

The processes of diffusion in solids are the subject of research in the field of solid state physics. Diffusion plays a decisive role in the growth processes of thin films, the formation of nanostructures on the surface of substrates and the sintering of ceramics, therefore, should be investigated.

Since in addition to the energy component of the process is still kinetic, it is necessary to consider their combination in experiments on the formation of nanocrystals. Regardless of energy stability conditions, some configurations may not be explained by this approach. For example, any volatility in crystals is energetically unprofitable, but nevertheless, arise during real experiments.

In this master's thesis will be considered the peculiarities of the formation of instabilities on the surfaces of materials with a diamond-like lattice on the example of the physical model on the sonnet of the Monte-Carlo method. The main objective is to study the conditions of the development of instabilities under different conditions, namely, on the main crystallographic planes, as well as the study of simultaneous evolution of several crystallographic planes. Detection of possible patterns between these factors plays a significant role in understanding the processes of evolution of nanocrystals.The result of the work is a computer program that surface phenomena in the nanoscale. Conditions for the formation of instabilities are obtained.

More than 100 experiments were conducted with a total data volume of 200 gigabytes. The report shows 20 pictures, pages -83, and literature -43.

Keywords: diffusion, ala-like lattice, numerical model.The result of the work is a computer program that describes the process of sintering nanoclusters. A reflection of the density of the structure formed from the sintering regime used.

ЗАВДАННЯ

на магістерську роботу студента

Стадніченка Григорія Олескандровича

1. Тема роботи: Нестійкості росту наноструктур із алмазоподібною граткою в дифузійному режимі.

Керівник роботи: д. фіз. мат. н., проф. Горшков В. М.

Затверджено наказом по університету від «02» листопада 2018р. №4064-с

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: за допомогою розробленого програмного забезпечення дослідити умови розвитку нестійкостей на поверхнях кристалу із алмазоподібною граткою.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити):

- Ознайомитися з процесом дифузії та будовою алмазоподібної гратки.
- Ознайомитися з чисельною моделю руху вільних та зв'язаних атомів.
- Розробити комп'ютерну программу для розрахунку динаміки росту наноструктур із алмазоподібною граткою.
- Порахувати тестову задачу, для перевірки чисельної моделі
- Порахувати задачу задачу із моделювання еволюції площини для декількох випадків.
- Побудувати графіки, які описують процеси конденсації.

5. **Перелік графічного матеріалу:** 14 малюнків

6. **Дата видачі завдання** 03.09.2018р.

Календарний план

Назва етапів виконання магістерської дисертації	**Строк виконання етапів роботи**	**Примітка**
Перегляд літературних джерел, складання огляду літератури	10.03.2019-25.03.2019	
Пошук і добір фактичних матеріалів, їх групування та систематизація	26.03.2019-31.03.2019	
Написання коду комп'ютерної програми для розрахунку розпаду вістря вольфрамових голок	01.04.2019-20.04.2019	
Запуск тестового експерименту задля перевірки створеної фізичної моделі	15.04.2019-19.04.2019	
Проведення основних експериментів зі спікання	20.04.2019-26.04.2019	
Обробка та візуалізація отриманих результатів	25.04.2019-03.05.2019	
Опис отриманих результатів	04.05.2019-14.05.2019	
Попередній захист	15.05.2019	
Корегування роботи відповідно до зауважень комісії	16.05.2019-19.05.2019	
Остаточний захист роботи	20.05.2019	

Магістр _____ Стадніченко Г.О.

(підпис) (ініціали, прізвище)

Керівник роботи _____ Горшков В. М.

(підпис) (ініціали, прізвище)

Реферат

Об’єкт дослідження: нанорозмірів.

Мета роботи: розробка програмного забезпечення для розрахунку динаміки конденсації із газової або рідкої фази, отримання результатів розрахунку.

Методи дослідження:

теоретичні: узагальнення даних стосовно теми дослідження на основі науково-методичної літератури та офіційних освітньо-наукових джерел;

практичні: написання коду програми для реалізації процесів конденсації із газової або рідкої фази.

Завдання дослідження:

- Ознайомитися з процесом дифузії та будовою алмазоподібної гратки.
- Ознайомитися з чисельною моделю руху вільних та зв'язаних атомів.
- Розробити комп'ютерну программу для розрахунку динаміки конденсації із газової або рідкої фази.
- Порахувати тестову задачу, щоб упевнитися в правильності роботи программи.
- Порахувати задачу для декількох випадків.

Результати та їх новизна

Розроблена комп’ютерна програма для розрахунку конденсації із газової або рідкої фази для речовин із алмазоподібною кристалічною граткою.

Отримана залежність наявності нестійкостей росту кристалів від концентрації частинок в газі/розчині.

Ступінь впровадження: робота є науковою базою для подальших досліджень.

Рекомендації щодо використання результатів роботи та область їх застосування: результати роботи можуть бути використані в мікро- та наноелектроніці, для виготовлення елементів методом гомоепітаксії.

Значення роботи та висновки: розроблена комп'ютерна модель нестійкого росту кристалів на площині.

Ключові слова: *Поверхнева дифузія, алмазоподібна гратка, чисельна модель, графік.*

В роботі наведено: використаної літератури - 29, сторінок: - 35, рисунків- 1

Abstract

Object of research: nanoscale surfaces

Purpose: the development of software for calculating the dynamics of condensation from the gas or liquid phase, obtaining the results of the calculation.

Research methods:

theoretical: generalization of data on the subject of research on the basis of scientific methodological literature and official educational-scientific sources;

Practical: writing code for the program to realize the condensation processes from the gas or liquid phase.

Objectives of the study: To familiarize with the process of diffusion and the structure of a diamond-shaped lattice.

• Look at the numerical model of the motion of free and bound atoms.

• Develop a computer program to calculate the gas or liquid phase condensation dynamics.

• Calculate the test task to ensure that the programs are working correctly.

• Calculate the task for several occasions.

Results and their: novelty A computer program for calculating condensation from gas or liquid phase for substances with diamond-like crystalline lattice has been developed. Dependence of the presence of instability of growth of crystals on the concentration of particles in gas / solution is obtained.

Degree of implementation: work is a scientific basis for further research.

Recommendations on the use of the results of work and the scope of their application: the results of work can be used in micro and nanoelectronics, for the manufacture of elements by the method of homoepitaxy.

Value of work and conclusions: a computer model of unstable growth of crystals on a plane is developed.

Keywords: surface diffusion, diamond-shaped lattice, numerical model, graph.

The paper contains: used literature – 29, pages: - 35, drawings - 1

ЗМІСТ

ВСТУП

Процеси дифузії у твердих тілах є предметом досліджень у галузі фізики твердого тіла. Дифузія відіграє визначальну роль у процесах росту тонких плівок, формуванні наноструктур на поверхні підкладок і спікання кераміки, тому має бути досліджена.

Оскільки окрім енергетичної складової процесу є іще кінетична, необхідно розглядати їх комбінацію в експериментах із формування нанокристалів. Не зважаючи на умови стійкості з точки зору енергії, деякі конфігурації можуть виявитись не пояснюваними за допомогою такого підходу. Так, наприклад, будь які нестійкості в кристалах є енергетично невигідними, але, тим не менш, виникають під час реальних експериментів.

У цій магістерський дисертації буде розглянуто особливості утворення нестійкостей на поверхнях матеріалів із алмазоподібною граткою на прикладі фізичної моделі на сонові методу Монте-Карло. Основною метою є дослідження умов розвитку нестійкостей за різних умов, а саме, на основних кристалографічних площинах, а також дослідження одночасної еволюції декількох кристалографічних площин. Виявлення можливих закономірностей між цими факторами грає визначну роль у розумінні процесів еволюції нанокристалів.

РОЗДІЛ І. Теоретичні відомості

1.1. Самодифузія

Самодифузія – це явище, що відбувається в однорідному газі за наявності градієнта концентрації і характеризується перенесенням односортних молекул. Величина самодифузії (А) визначається співвідношенням концентрацій: $A= n/n_{0,}$, де n_0 – рівноважна концентрація молекул у газі, n – концентрація.

Для прикладу, розглянемо експеримент: є об’єм, розділений навпіл перегородкою. Одна половина заповнена газом, а інша радіоактивним ізотопом цього ж газу. Позначимо концентрації цих молекул як n_1 і n_2 відповідно, і допоки система знаходиться в цьому стані покладемо: $n_1 = n_2 = n$.

Після того, як перегородку буде прибрано, молекули почнуть перемішуватись в наслідок теплового руху і величина концентрації молекул кожного ізотопу буде змінюватись вздовж напрямку, перпендикулярному площині перетинки (приймемо це за вісь x), при тому:

$$n_1(x) + n_2(x) = n;$$

$$-\frac{dn_1}{dx} = \frac{dn_2}{dx}.$$

Виділимо в об’ємі площадку площею dS, перпендикулярну вісі x і аналогічно до випадку з теплопровідністю, запишемо вираз для маси кожного ізотопу, перенесеної через цю площадку. Це й закон зветься законом Фіка, є підтвердженим експериментально і виглядає наступним чином:

$$dM_1 = -Dm\frac{dn_1}{dx}dS = -D\frac{d\rho_1}{dx}dS$$

І, як наслідок, потік молекул одного ізотопу через площадку:

$$dN_1 = \frac{dM_1}{m} = -D\frac{dn_1}{dx}dS,$$

де D це коефіцієнт дифузії і визначається дифузійним потоком, при одиничному градієнті концентрації.

1.2. Вільна поверхнева енергія

Вільна енергія є однією із основних термодинамічних величин і позначається $\mathcal{F}$. Однією із властивостей рівноважної конфігурації є мінімальна вільна енергія. Одначе, завжди варто спочатку визначити додаткові умови рівноважного стану, оскільки вільна енергія може мати локальні мінімуми в певних конфігураціях, і досліджуваний ефект є прикладом такого мінімуму.

В системі із двома фазами, розділеними поверхнею розподілу вільну енергію можна визначити як:

$$\mathcal{F} = f_A(T)\Omega_A + f_B(T)\Omega_B + \mathcal{F}_S$$

, де $f_i(T)$ це густина енергії відповідної фази A, B; Ω_i це їх об'єм, а

$\mathcal{F}_S$ це вільна енергія поверхні розділу фаз. Вона може бути записана як інтеграл по поверхні:

$$\mathcal{F}_S = \int_{\partial\Omega} dA\gamma(n,T)$$

, де $\gamma(n,T)$ це вільна енергія на одиницю площі. Форма поверхні визначається мінімумом поверхневої вільної енергії. Простим прикладом може служити крапля: так як рідини є ізотропними, то мінімум енергії визначатиметься мінімальною площею поверхні, що приводить до сферичної форми. В твердих тілах ця форма складається із тих самих площин, що і конфігурація Вульфа для кристалічної гратки речовини, із якої складається тіло. Вартим уваги є те, що співвідношення площ граней, утворених площинами залежить від конкретної речовини і, власне, вищевказаним рівнянням для вільної поверхневої енергії.

1.3. Модель

За низьких температур кристали мають достатньо мало дефектів, щоб ними знехтувати, і представити кристал у вигляді масиву цілих чисел, де кожну частинку представлено функцією із одним значенням. Гамільтоніан для подібних систем може бути представлений як сума енергій всіх частинок, що знаходяться на поверхні:

$$H = \sum_{\langle i,j \rangle} K\left|h_i - h_j\right| + H_0$$

, де h_i це висота поверхні i, і сума йде по всіх сусідніх парах $\langle i, j \rangle$. H_0 є енергією пласкої поверхні h_i, що є константою і буде прийнята за нуль.

Така модель не покриває динамічні процеси, тож процеси міграції, випаровування та конденсації повинні бути додані окремо. Найлегшим способом виконати цю задачу буде дозволити частинкам на поверхні перестрибувати на сусідні вакансії, а також відриватись від поверхні. При чому, стрибки будуть миттєвими.

Однак, необхідно визначити ймовірності цих стрибків. Ймовірність стрибку залежить від глибини потенційної ями, в якій знаходиться частинка на даний момент, а ймовірність переміщення в конкретний сусідній вузол кристалічної гратки визначається різницею у глибині потенційних ям: поточної і цільової. Легко бачити, що ймовірнісна частина процесу підкорюється розподілу Больцмана:

$$P(i) \sim \exp(-\beta E(i))$$

1.4. Метод Монте-Карло

Моделі на основі методу Монте-Карло є достатньо поширеним інструментом для побудови моделей в багатьох галузях фізики. Основна ідея полягає у тому, щоб створити серію станів системи з елементом випадковості, а не слідувати по прямій часу еволюції, що тільки збільшить фазовий простір системи. Послідовність конфігурацій має бути побудована в такий спосіб, щоб кожна конфігурація мала свою статистичну вагу. На практиці, такі моделі будуються за допомогою задання ймовірностей переходів між конфігураціями і послідуючим оновленням конфігурації.

Легко бачити, що розподіл Больцмана із попереднього розділу може бути утворений правильним вибором переходів, які задовольняють умові рівноваги системи. Після кожного переходу час системи інкрементується на τ, що є характерним часом системи.

Важливим аспектом у визначенні ймовірностей перетворень є визначення найшвидшого процесу в системі. Так, як метод Монте-Карло будується навколо головного циклу, який виконується до досягнення характерного результату (наприклад, утворення кластеру певної форми, проходження достатньої кількості часу). Кожен крок циклу зветься тіком. Найшвидший процес має ймовірність перетворення 1. Ймовірність всіх інших перетворень вимірюється відносно найшвидшого і обов'язково не перевищує 1. В противному випадку це свідчить про те, що найшвидший процес було обрано не вірно.

Основний цикл такої моделі зазвичай складається із наступних кроків:

1) Випадковим чином обирається перетворення і обраховується його ймовірність відносно найшвидшого процесу в системі.
2) Генерується випадкове число $N \in [0; 1]$ і порівнюється із ймовірністю перетворення. Якщо $N < P_i$, де P_i це ймовірність перетворення,

перетворення виконується і зміни зберігаються в моделі. В противному випадку не відбувається нічого.

3) Початок нової ітерації.

Цей метод організації обчислень є достатньо ефективним з точки зору програмування, оскільки оцінка доцільності кроку відбувається до його виконання, а система ймовірностей вірно балансує у швидкості різні процеси. Варто зазначити що кількість відкинутих кроків зазвичай сильно перевищує кількість виконаних, тому швидкодії оцінки доцільності (обрахуванню ймовірності та отриманню випадкового числа) варто приділити особливу увагу.

Випадку варіативності процесу перетворення, для нього можна побудувати граф ймовірностей, де корінь має ймовірність самого перетворення, а кожен наступний набір дочірніх нод зберігає інформацію тільки про співвідношення між ймовірностями перетворень на цьому рівні дерева. Таким чином для кожного складного перетворення утворюється простий і ефективний алгоритм визначення, що не потребує значних ресурсів і додаткового аналізу. Особливо ефективним в такому випадку є бінарне дерево, оскільки вибір гілки перетворення відбувається за час O(1) а не O(2n), де n – кількість варіантів перетворення на рівні, але треба пам’ятати, що вид дерева визначається алгоритмом перетворення.

1.5. Опис реалізації програми

Так, як програму побудовано на методі Монте-Карло, то перше, що необхідно зробити, це визначити як саме виглядатиме конфігурація системи. В даному випадку, об’єктом моделювання є кристалічна гратка твердого тіла і простір навколо неї, тож буде зручно використовувати для моделювання простору трьохвимірний масив. Також необхідно «запакувати» елементарну комірку цільової гратки в цей масив і визначити алгоритм знаходження сусідів а також обмежувальну умову, для того, щоб частинки не могли проникати між вузлами кристалічної гратки. Упаковка вдасться доволі легко, якщо звернути увагу на будову комірки кристалічної гратки на малюнку. На ньому видно, що під кожен вузол вже виділено кубічний об’єм простору.

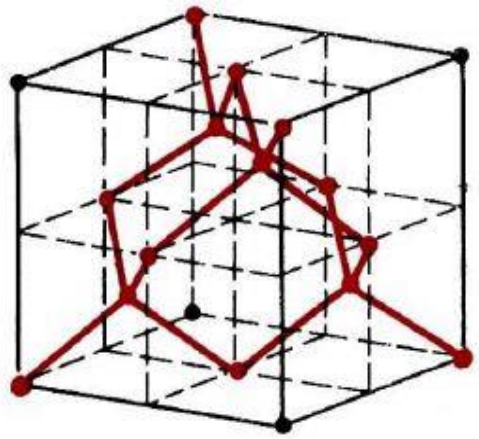


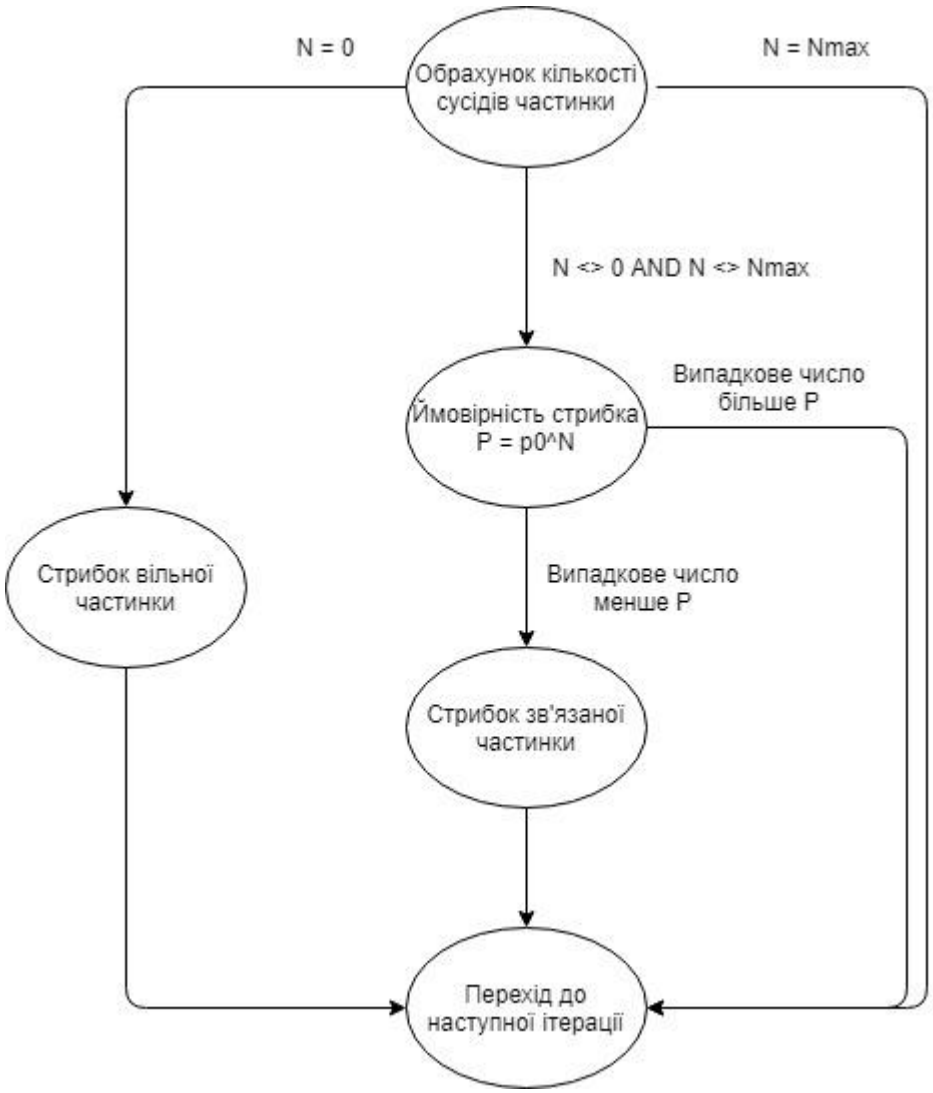
Рис. 1.5.1 Елементарна комірка алмазоподібної гратки.

Другим етапом буде визначення найшвидшого процесу в системі. Їм безперечно стане рух вільних частинок, оскільки молекули в газі або рідині не мають жорсткого зв’язку між собою, і, як наслідок, рухаються максимально часто.

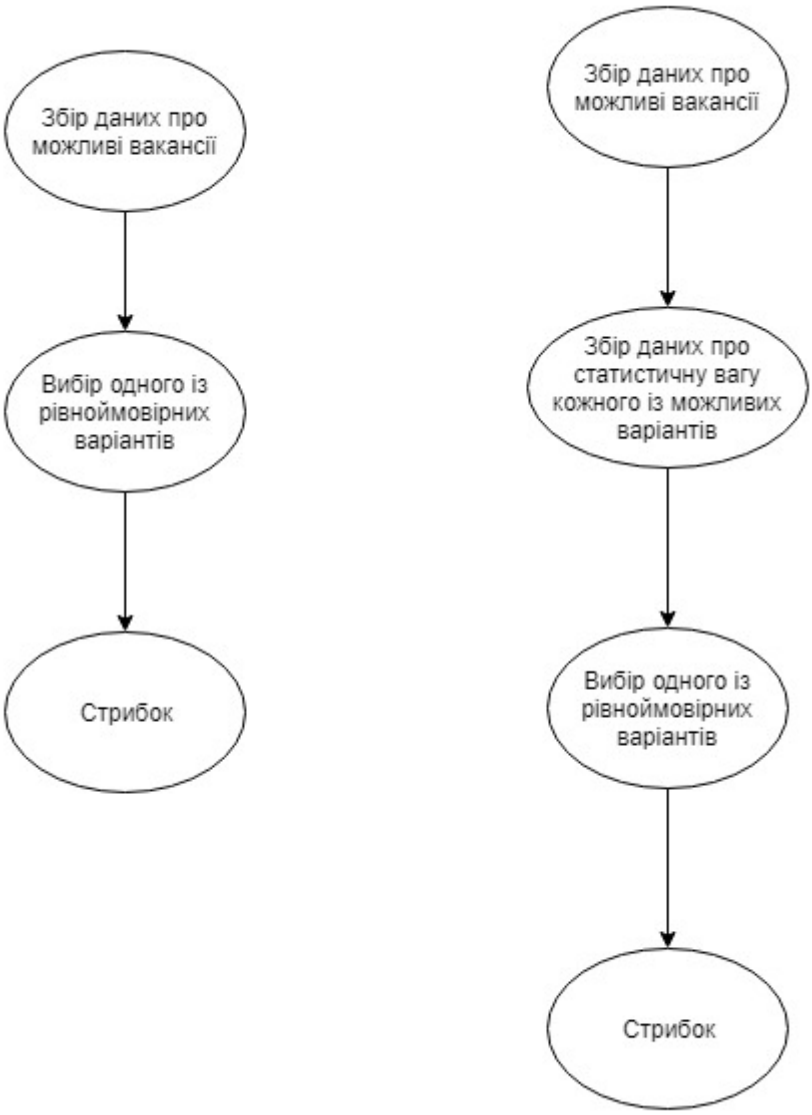
Наступним стане визначення графа ймовірностей для перетворень. Не зважаючи не те, що логічно вид перетворення тільки один, стрибок кожної окремої частинки є окремим перетворенням і один Монте-Карло крок складатиметься із N спроб на стрибок випадкових частинок, де N це кількість частинок в системі. Так як вид перетворення тільки один (стрибок частинки), то і граф буде один. Згідно із моделлю, описаною вище, ймовірність руху частинки обернено пропорційна глибині потенційної ями, в якій вона

знаходиться. Для оптимізації процесу зведемо цю частину алгоритму до алгоритму на малюнку [ ].

Як відомо, глибина потенційної ями, в якій знаходиться частинка, прямо залежить від кількості зв’язаних із нею частинок, тому найпершим кроком треба визначити їх кількість. Із отриманої кількості легко визначити, чи частинка може кудись стрибнути взагалі. Для цього кількість сусідів просто порівнюється із максимальною, коли всі сусідні місця зайняті. Прості математичні перетворення було опущено в наданій блок-схемі.



Також, варто зазначити, що «Стрибок вільної частинки» і «Стрибок зв’язаної частинки» не є простими операціями і їх також можна представити у вигляді вкладених графів.



Варто звернути увагу на те, що ймовірність руху вільної частинки рівно ймовірна для усіх напрямків на відміну від шансів для зв’язаної.

РОЗДІЛ ІІ. Проведені експерименти

2.1. Конфігурація Вульфа

Одним із ключових критеріїв працездатності розробленої моделі є здатність відтворити поведінку речовини без зовнішнього впливу. Якщо результати такого моделювання не повторюватимуть реальної поведінки, то модель однозначно побудовано невірно.

Для даного випадку було обрано об’єм кубічної форми із ребром у 100 сталих кристалічної гратки. В середині об’єму розташовано кластер кулеподібної форми.

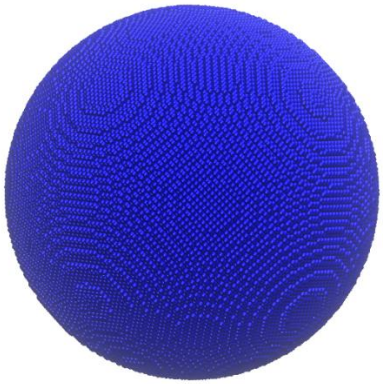


Рис. 2.1.1 Початкова форма кристалу

При значенні температурного коефіцієнта $\alpha = 2.8$ процес починається. Таке значення параметра відповідає низьким температурам.

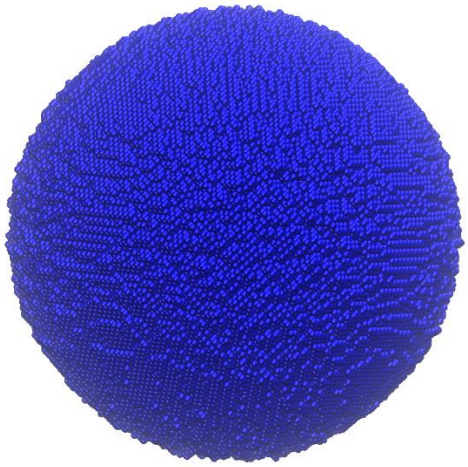


Рис. 2.1.2 1000 крок

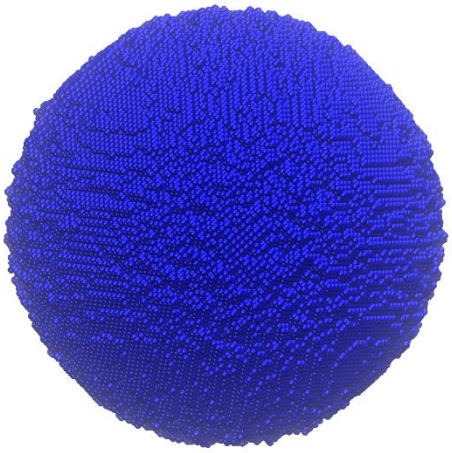


Рис. 2.1.3 2000 крок

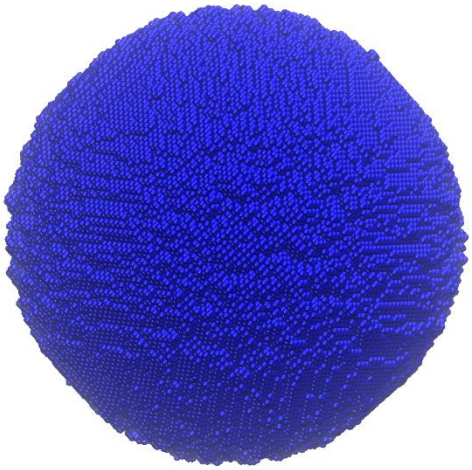


Рис. 2.1.4 5000 крок

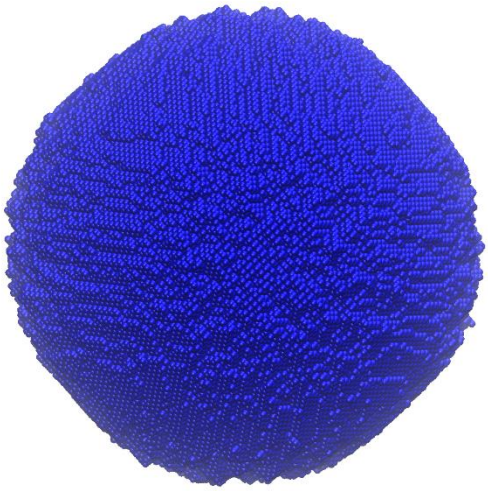


Рис. 2.1.5 10000 крок

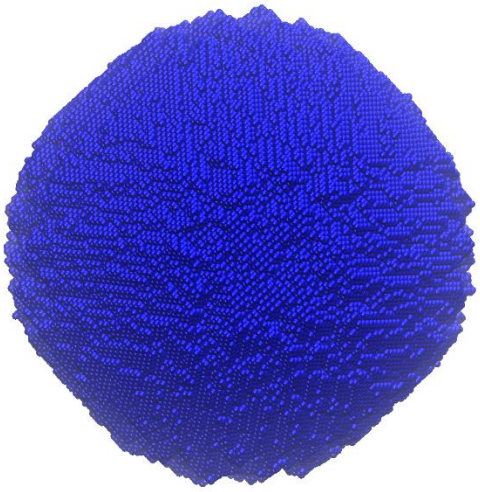


Рис. 2.1.6 20000 крок

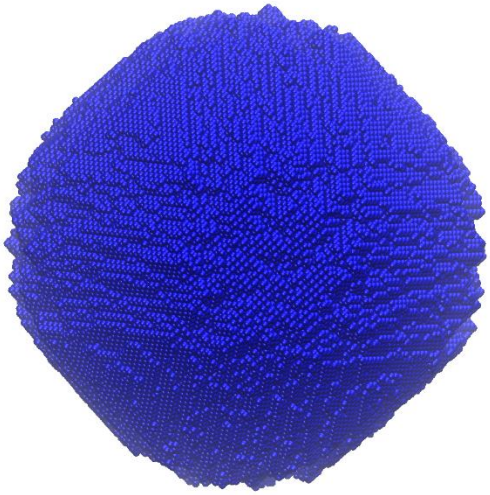


Рис. 2.1.7 50000 крок

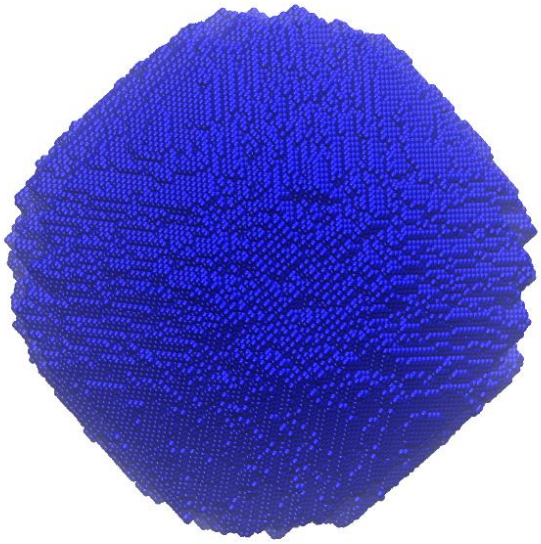


Рис. 2.1.8 50000 крок

Так, як процес відбувався при середніх температурах, грані кристалу дещо ступінчасті, але попри це добре видно всі грані конфігурації Вульфа для алмазоподібної кристалічної гратки. Такий результат дозволяє продовжити дослідження із використанням розробленої моделі.

2.2. Еволюція поверхні за наявності виключно поверхневої дифузії

Для дослідження впливів різноманітних складових дифузії на результат процесу є сенс провести експеримент із еволюцією площини за наявності тільки поверхневої дифузії. Суть експериментів полягає у тому, щоб ізолювати кожну зі складових процесу і визначити, до якого результату вона приведе. В даному випадку, необхідно перевірити на скільки стійкою є площина виду (1; 0; 0) за низьких температур.

Розглянемо структурні особливості цього виду площин. Кожен із атомів, який є складовою поверхні площини (1; 1; 1) зв'язаний із трьома сусідами і відтак є міцно закріпленим, через те, що знаходиться у глибокій потенційній ямі. Будь який атом, що знаходиться на цій площині матиме тільки одного сусіда, та більше, будь який із сусідніх вузлів кристалічної гратки матиме таку саму, незначну, глибину. Відтак, зміна свого розташування для такого атома буде значно більш ймовірною, ніж для атома самої поверхні, і він активно пересуватиметься по ній. У вузлах гратки, що знаходяться на перетині площин виду (1; 0; 0) і (1; 1; 1) глибина потенційної ями буде більшою, ніж на поверхні площини (1; 1; 1), відтак, атоми, що «скотяться» із неї, швидше за все, стануть її частиною. Для перевірки цих гіпотез проведемо три експерименти:

1) Конфігурація Вульфа за відсутності випаровування
2) Еволюція площини виду (1; 0; 0).
3) Еволюція площини виду (1; 1; 0).
4) Еволюція площини виду (1; 1; 1).

Ключовою метою проведення кожного із цих дослідів є поглиблення розуміння впливу процесів поверхневої дифузії на розвиток нестійкостей.

1) Конфігурація Вульфа за відсутності випаровування

Як і в попередньому експерименті, процес починається із кулеподібного кластеру, але цього разу, за низьких температур. Основною метою цього досліду є розуміння того, які саме процеси найбільше впливають на формування граней конфігурації Вульфа. Можливо, не зважаючи на однакову поверхневу енергію, деякі грані все ж матимуть перевагу над іншими. Наприклад, у відповідності із процесами, описаними вище.

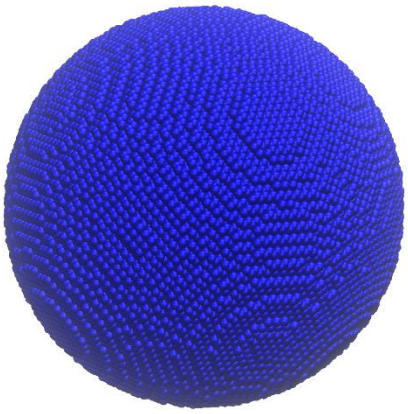


Рис. 2.2.1.1 Початкова форма

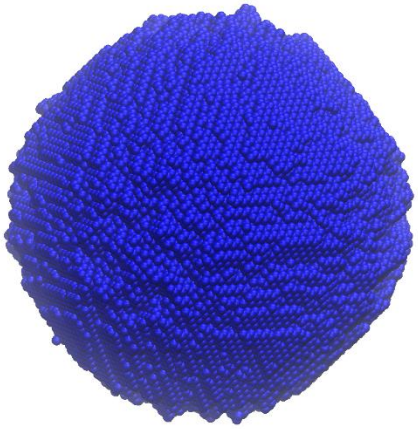


Рис. 2.2.1.2. Крок 10000

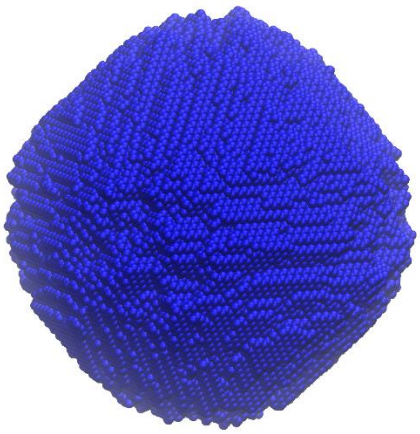


Рис. 2.2.1.3 Крок 20000

Грані виду (1; 1; 1) достатньо чітко вимальовуються після 200000 кроків Монте-Карло.

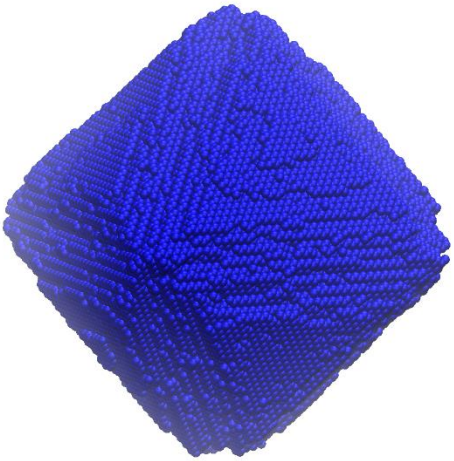


Рис. 2.2.1.4 Крок 50000

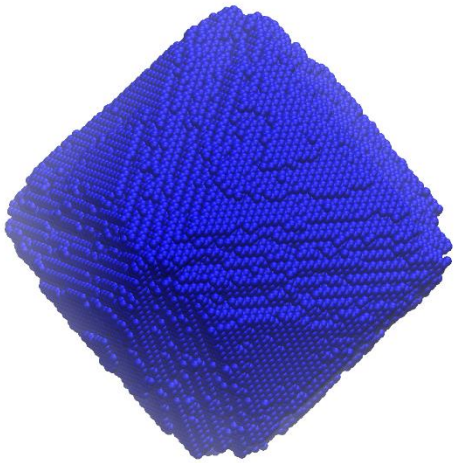


Рис. 2.2.1.5 Крок 100000

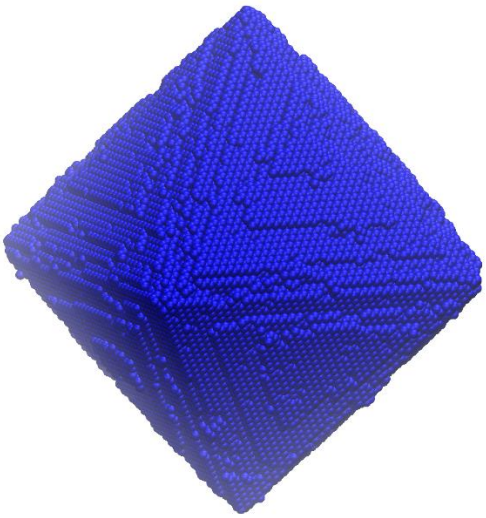


Рис. 2.2.1.6 Крок 500000

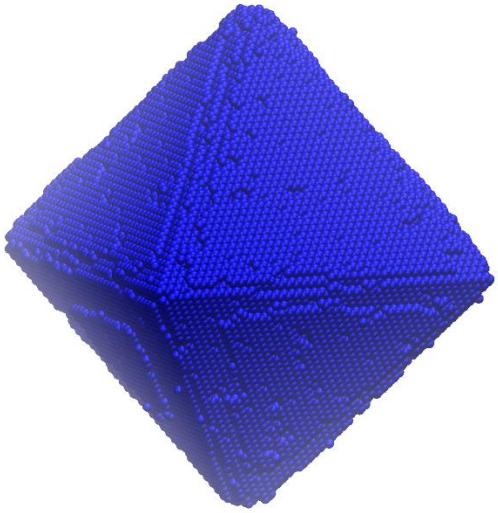


Рис. 2.2.1.7 Крок 1300000

Пройшло 1300000 кроків Монте-Карло, і на даний момент можна вже впевнено судити про те, які грані домінують за відсутності випаровування і низьких температур. Це грані виду (1; 1; 1). Результатом експерименту є октаедр, що співпадає із формою природнього алмазу, що дозволяє проводити паралелі між умовами формування природніх кристалів і параметрами фізичної моделі.

2) Еволюція площини виду (1; 0; 0)

Процес починається із кристалу речовини, плоскої форми, поверхня якого має нормаль (1; 0; 0).

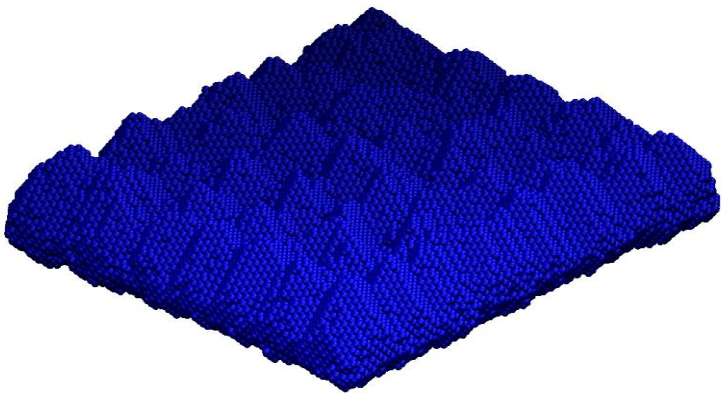


Рис. 2.2.2.1 Нестійкості.

Як видно, на поверхні площини утворились чималі пірамідальні нерівності. Не зважаючи на те, що поверхнева енергія в такому стані більша, ніж у початковому, він є стабільним саме через наявність динамічних процесів.

3) Еволюція площини виду (1; 1; 0)

Процес починається із кристалу речовини, плоскої форми, поверхня якого має нормаль (1; 1; 0).

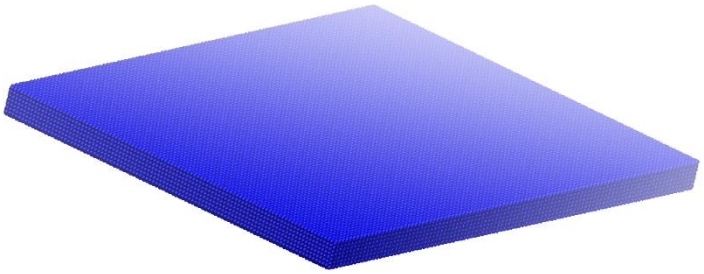


Рис. 2.2.3.1 Початкова конфігурація

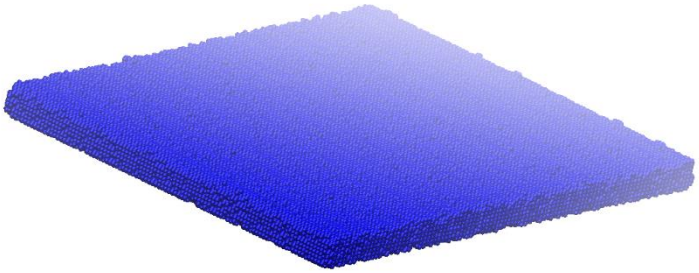


Рис. 2.2.3.2 10000 крок

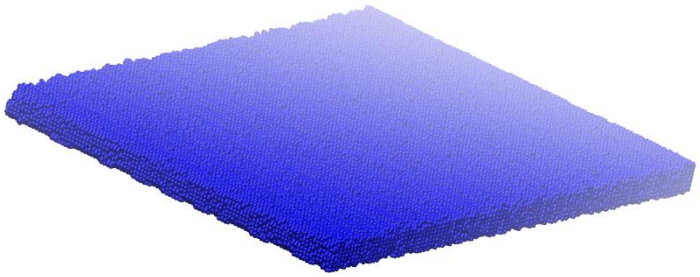


Рис. 2.2.3.3 20000 крок

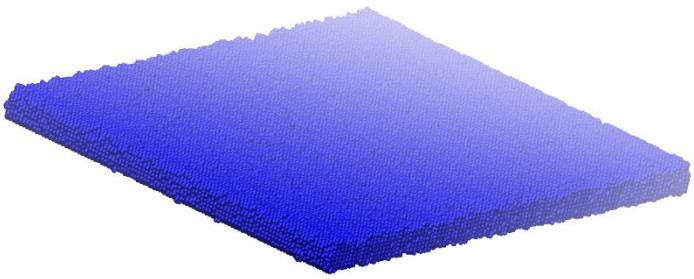


Рис. 2.2.3.4 35000 крок

Як видно, поверхня тільки набула ледь нерівного вигляду, при тому, висота нерівностей не перевищує один атом ,носить виключно випадковий характер.

4) Еволюція площини виду (1; 1; 1)

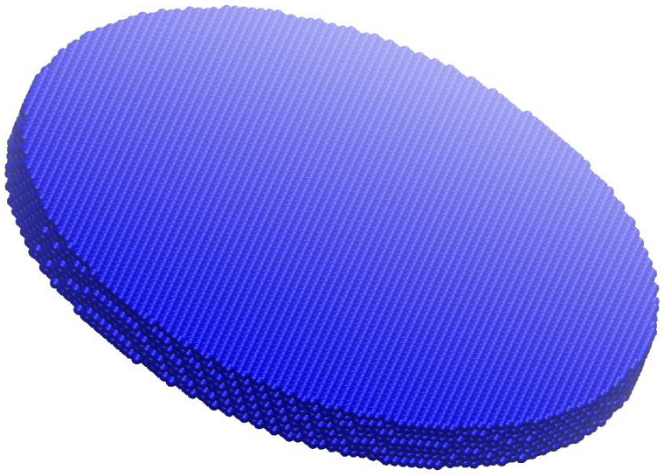


Рис. 2.2.4.1 Початкова конфігурація

Процес починається із кристалу речовини, плоскої форми, поверхня якого має нормаль (1; 1; 1).

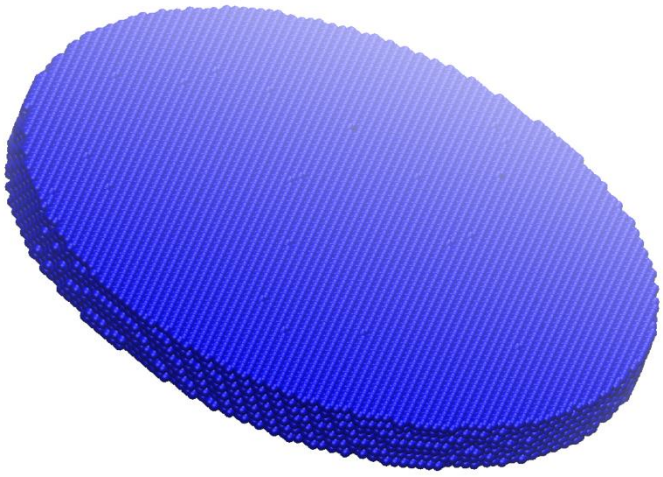


Рис. 2.2.4.2 10000 крок

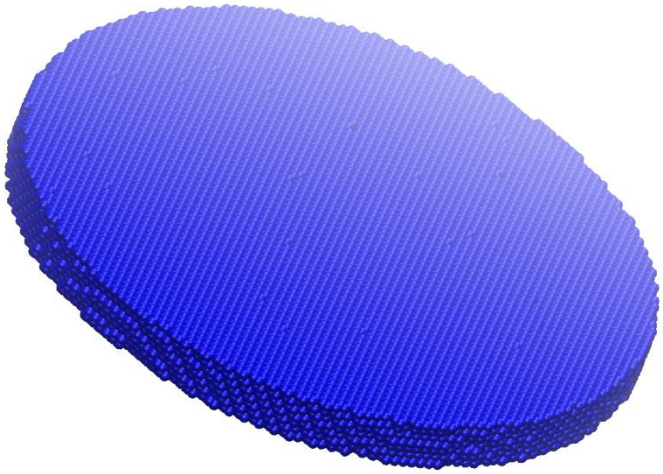


Рис. 2.2.4.3 20000 крок

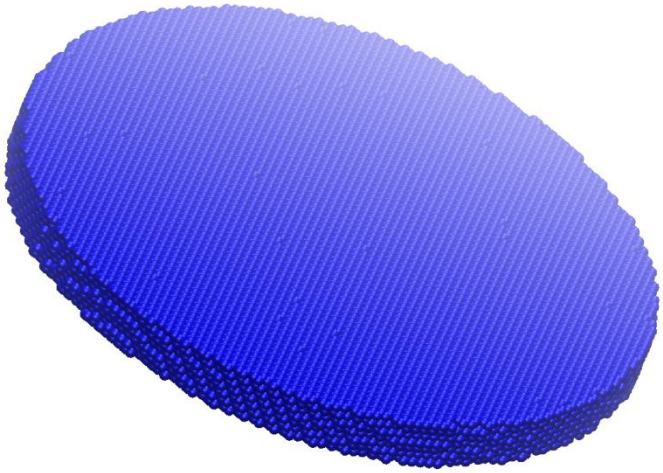


Рис. 2.2.4.2 38000 крок

Як видно, поверхня тільки набула ледь нерівного вигляду, при тому, висота нерівностей не перевищує один атом ,носить виключно випадковий характер і ефект носить назву Thermal Surface Roughening.

2.3. Конденсація на площину при емісії вільних атомів у систему

По аналогії із розділом про ізоляцію дифузійних процесів на одній поверхні кристалу, варто провести ряд експериментів із вираженим впливом не поверхневої дифузії. Так, як виключити процес поверхневої дифузії, при тому зберігши процес сублімації і конденсації в реальності не є можливим, то наступні експерименти триватимуть за вищих температур і із емісією вільних атомів у систему, де вони конденсуватимуться на плоскій підкладці із нерухомих атомів. Так, як нестійкості було виявлено тільки для площин виду (1; 0; 0), то і експеримент буде проведено тільки для такої площини.

В якості початкової конфігурації буде площина. Для наочності, на картинках її не було відображено для того, щоб атоми на ній було добре видно.

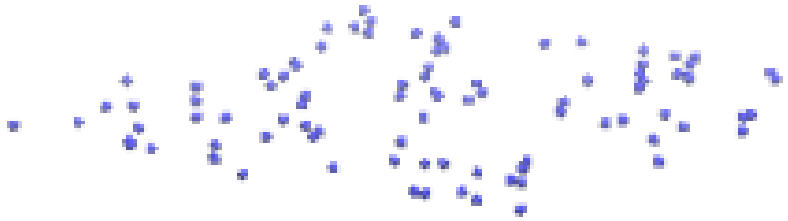


Рис. 2.3.1 Крок 100000

Рис. 2.3.2 Крок 200000

Рис. 2.3.3 Крок 500000

На цій серії картинок видно, як на поверхні осідає достатньо атомів для того, щоб почали утворюватись кластери. Цей процес триватиме і надалі.

Рис. 2.3.4 5000000 крок

На малюнку видно повністю сформовані пірамідальні структури. Так як до цього було виявлено залежність між відстанню між піками структур та концентрацію, інтерес представляє динаміка розвитку подій у випадку сильного збільшення концентрації газу. Наступні картинки ілюструють саме цей розвиток подій.

Рис. 2.3.5 Крок 5100000

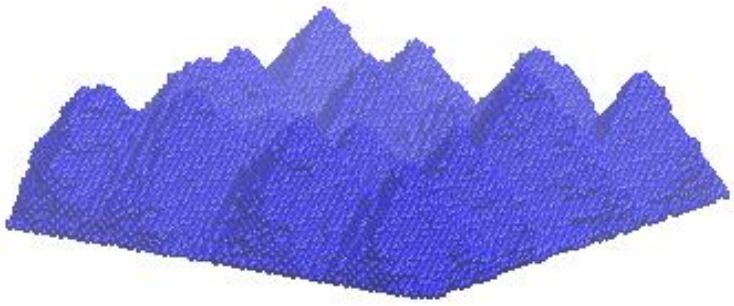


Рис. 2.3.6 Крок 5200000

Рис. 2.3.7 Крок 5300000

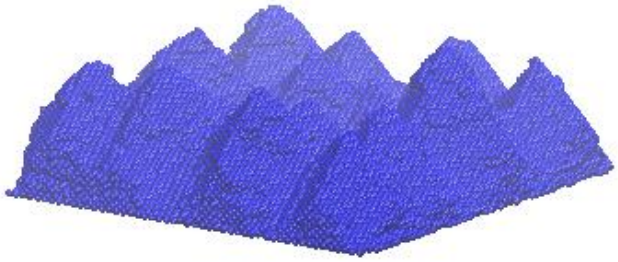


Рис. 2.3.8 Крок 5400000

Рис. 2.3.9 Крок 5500000

Рис. 2.3.10 Крок 5600000

Вже на цьому етапі видно порушення пірамідальної форми утворень і їх витягування.

Рис. 2.3.11 Крок 5700000

Рис. 2.3.12 Крок 5800000

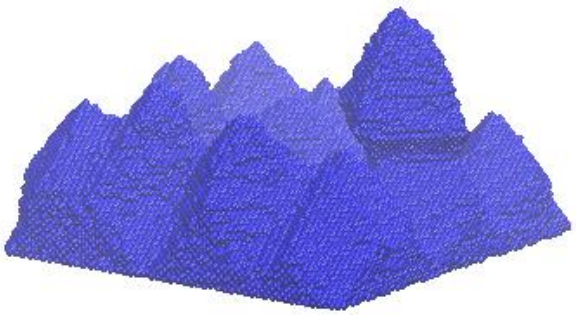


Рис. 2.3.13 Крок 5900000

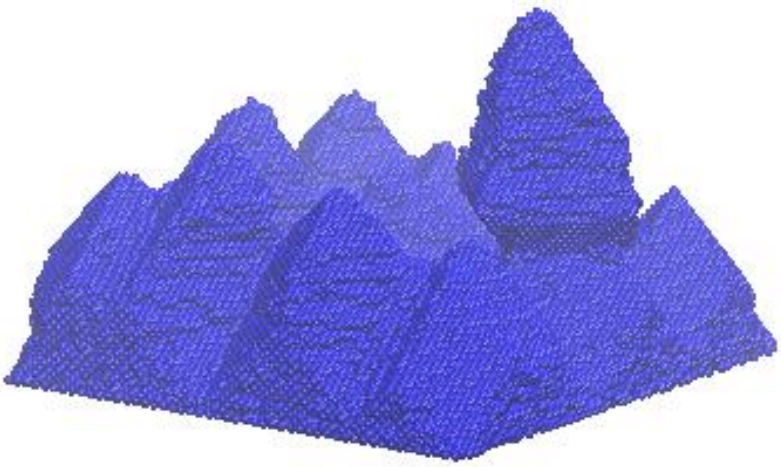


Рис. 2.3.14 Крок 6000000

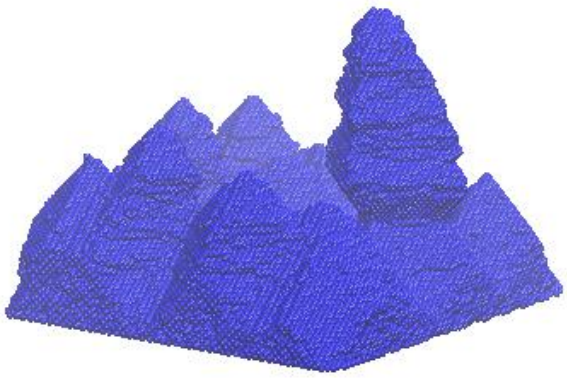


Рис. 2.3.15 Крок 6100000

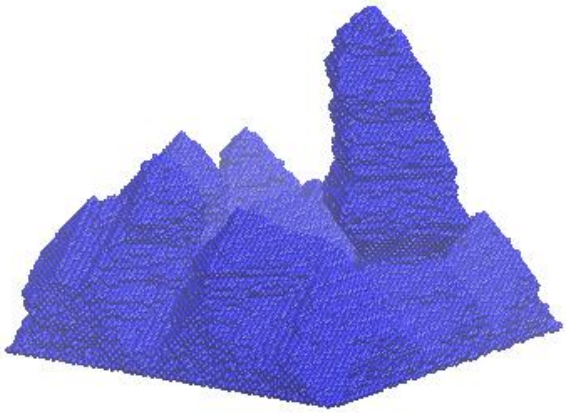


Рис. 2.3.16 Крок 6200000

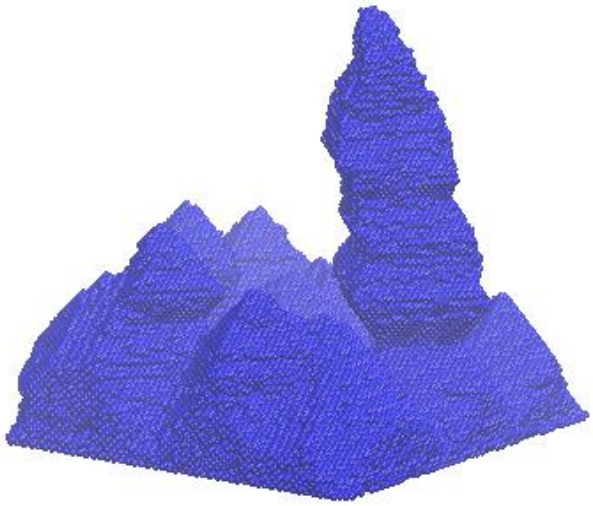


Рис. 2.3.17 Крок 6300000

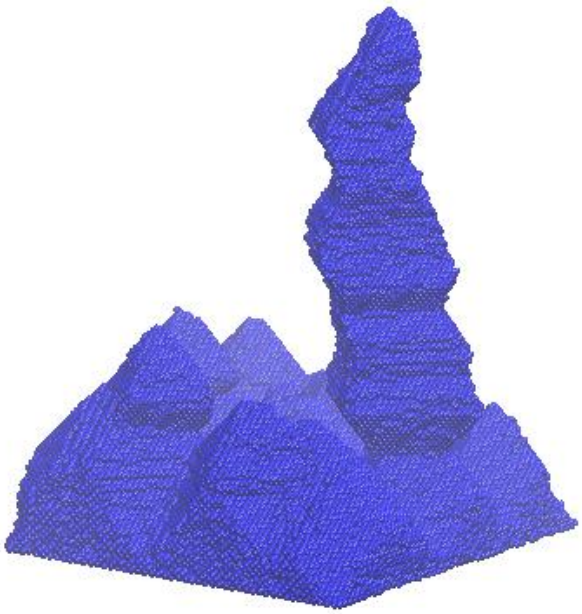


Рис. 2.3.18 Крок 6400000

Видно, що утворилось декілька витягнутих структур, одна із яких особливо помітна. Пояснити це можна розглянувши процес утворення кристалів. Кристали утворюються у випадку, коли концентрація частинок перевищує граничне значення. Між випадковим чином утвореними кластерами відбувається обмін частинками. При чому із більших кластерів випаровування відбувається менш активно, ніж із великих і таким чином відбувається наступна еволюція системи: за рахунок хаотичного обміну частинками великі кластери поглинають менші аж допоки не залишиться один кластер. За наявності емісії частинок в систему знайдеться така усталена конфігурація із N кластерів, що обмін частинками між ними буде меншим за обмін частинками із середовищем і всі кластери ростимуть. Однак, за рахунок того, що частинки прибувають зверху, вищі кластери матимуть більший приток частинок за рахунок більшого потоку частинок через їх поверхню. Більший потік зумовлений двома факторами: більшою площею, і більшим перепадом концентрацій. Таким чином, деякі кластери виявляться «лідерами» і в цьому процесі, що, власне, і спостерігається в експерименті.

ВИСНОВКИ

В даній роботі було поставлено за мету дослідити умови розвитку нестійкостей для алмазоподібної кристалічної гратки а також їх особливості. Для виконання задачі було обрано метод чисельного моделювання Монте-Карло, а також набір програмних інструментів, таких як Java і VMD, перший із яких слугував для реалізації моделі, а другий для візуалізації результатів.

Результати основних етапів дослідження:

1) Визначення основних кристалографічних площин для кристалічної гратки. Це було зроблено на прикладі експерименту із утворенням конфігурації Вульфа, яка і складається із таких площин. Результатом були площини трьох видів, напрямні вектори яких можна записати у загальних виглядах: (1; 1; 1), (1; 1; 0) і (1; 0; 0).

2) Визначення кінетики утворення кожної із площин. Не зважаючи на те, що на кожній із поверхонь енергія однакова, розглядається динамічний процес, а стало бути, частинки в різних площинах взаємодіють між собою, і також, можуть розвинутись нестійкості. Експеримент із утворенням конфігурації Вульфа без участі випаровування показав, що в динамічному процесі за низьких температур і високих тисків, площини вигляду (1; 1; 1) домінують в розвитку над іншими кристалографічними площинами.

3) Дослідження кожної площини на стійкість дало такі результати: нестійкими є тільки площини виду (1; 0; 0), які, як показує попередній експеримент, підтримуються за рахунок випаровування із них частинок, бо постійно існує приток на них нових частинок із сусідніх граней, і у випадку із повільним або відсутнім випаровуванням цей дрейф не дає утворитись площинам видів (1; 1; 0) і (1; 0; 0).

4) Розвиток нестійкостей на площинах виду (1; 0; 0) дав основу для постановки експерименту із конденсацією речовини на таку площину. В результаті було виявлено залежність між середньою відстанню між утвореними кластерами на поверхні і потоком речовини крізь неї. Як видно, при сильному збільшенні потоку, пірамідальні утворення виростають у колоноподібні, що є повтореним реальним експериментом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. D. T. Robb and V. Privman, Langmuir 24, 26 (2008); V. Privman, Ann. NY Acad. Sci., in print (arxiv.org/abs/0707.2595) and J. Optoelectr. Adv. Mater., in print (arxiv.org/abs/0806.4644).
2. A.A. Chumak and A.A. Tarasenko, Diffusion and density fluctuations of atoms adsorbed on solid surface, Surface Science, 91 (1980) 694-706.
3. А.А. Тарасенко, П.М. Томчук, А.А. Чумак, Флуктуации в объеме и на поверхности твердых тел, “Наукова думка”, 251 с., Киев, 1992.
4. C. M. Wang, D. R. Baer, J. E. Amonette, M. H. Engelhard, Y. Qiang and J. Antony, Nanotechn. 18, 255603 (2007).
5. J. Pan and R. Huang, Key Eng. Mater., 2008, pp.368–372.
6. Lord Rayleigh, On the instability of jets, Proc. London Math. Soc. 10, 4 (1878). https://doi.org/10.1112/plms/s1-10.1.4
7. S Karim, M E Toimil-Molares, A G Balogh, W Ensinger, T WCornelius, E UKhan and R Neumann, Morphological evolution of Au nanowires controlled by Rayleigh instability, Nanotechnology 17 (2006) 5954–5959, doi:10.1088/0957-4484/17/24/009.
8. C. Bréchignac, Ph. Cahuzac, F. Carlier, C. Colliex, J. Leroux, A. Masson, B. Yoon, and Uzi Landman, Instability Driven Fragmentation of Nanoscale Fractal Islands, Phys. Rev. Lett. 88, NUMBER 19, 13 MAY 2002, doi: 10.1103/PhysRevLett.88.196103.
9. Donghong Min and Harris Wong, **Rayleigh’s instability of Lennard-Jones liquid nanothreads simulated by molecular Dynamics,** Phys. Fluids 18, 024103 (2006); doi: http://dx.doi.org/10.1063/1.2173620.
10. Amlan Dutta, Swastika Chatterjee, A. K. Raychaudhuri, Amitava Moitra, and T. Saha-Dasgupta**, In-silico investigation of Rayleigh instability in ultra-thin copper nanowire in premelting regime,** J. Appl. Phys. **115**, 244303 (2014); http://dx.doi.org/10.1063/1.4885044.

11. M. Tjahjadi, H. A. Stone and J. M. Ottino, **Satellite and subsatellite formation in capillary Breakup,** J. Fluid Mech. (1992), v. 243, pp. 297-317, doi: https://doi.org/10.1017/S0022112092002738.

12. H. A. Stone**, Dynamics of drop deformation and breakup in viscous fluids,** Annual Review of Fluid Mechanics, (1994), vol. 26, pp.65-102, DOI:10.1146/annurev.fl.26.010194.000433.

13. V. N. Gorshkov and D. V. Mozyrskii, **Self-excitation of short-wave structures and breakup into droplets in bounded liquid filaments**, Technical Physics, Volume 41, Issue 10, October 1996, Pages 980-985.

14. R. M. S. M. Schulkes, The contraction of liquid filaments, J. Fluid Mech., Volume 309 February 1996, pp. 277-300, doi: https://doi.org/10.1017/S0022112096001632.

15. Jens Eggers, **Nonlinear dynamics and breakup of free-surface flows**, Reviews of Modern Physics, Vol. 69, No. 3, July 1997, pp. 865-929.

16. V. N. Gorshkov and M. G. Chaban, **Nonlinear electrohydrodynamic phenomena and droplet generation in charged jets of conducting liquid**, Technical Physics, v. 44, Number 11 November 1999, pp. 1259-1266, doi:10.1134/1.1259506.

17. A.A. Castrejron-Pita, J.R. Castrejron-Pita, and I.M. Hutchings, **The breakup of liquid filaments**, Phys. Rev. Lett. 108, 074506 (2012).

18. C.-H. Zhang, F. Kassubek, and C. A. Stafford, **Surface fluctuations and the stability of metal nanowires**, Phys. Rev. B 68, (2003).

19. Sevonkaev, I.; Goia, D. V.; Matijevic, E. J. Colloid Interface Sci. 2008, 317, 130.

20. Halaciuga, I.; Goia, D. V. J. Mater. Res. 2008, 23, 1776.

21. Goia, D. V.; Jitianu, M. J. Colloid Interface Sci. 2007, 309, 78.

22. Jitianu, M.; Kleisinger, R.; Lopez, M.; Goia, D. V. J. New Mater. Electrochem. Syst. 2007, 10, 67.

23. Wang, C. M.; Baer, D. R.; Amonette, J. E.; Engelhard, M. H.; Qiang, Y.; Antony, J. Nanotechnology 2007, 18, 255603.

24. Matijevic, E. Colloid J. 2007, 69, 29.

25. Uskokovic, V.; Matijevic, E. J. Colloid Interface Sci. 2007, 315, 500.

26. Nair, P. S.; Fritz, K. P.; Scholes, G. D. Small 2007, 3, 481.

27. Embden, van J.; Jasieniak, J.; Gomez, D. E.; Mulvaney, P.; Giersig, M. Aust. J. Chem. 2007, 60, 457.

28. Yin, Y. D.; Erdonmez, C.; Aloni, S.; Alivisatos, A. P. J. Am. Chem. Soc. 2006, 128, 12671.

29. Zeng, H. B.; Liu, P. S.; Cai, W. P.; Cao, X. L.; Yang, S. K. Cryst. Growth Des. 2007, 7, 1092.

ДОДАТКИ

Основний код програми.

1. Main

1.1 Main

```
package application;

import Helpers.Point;
import engine.*;

public class Main {

    public static void main(String[] args) {

        Grid grid = new Grid(200, 200, 50);
        GridHelper helper = new GridHelper();
        helper.grid = grid;

        Point po = new Point(100, 100, 200);
        double radius = 60;


        for(int i = 0; i < grid.dimX; i++) {
            for(int j = 0; j < grid.dimY; j++) {
                for(int k = 0; k < grid.dimZ; k++) {
                    if(
                        k < grid.dimZ / 2
                    )
                    {
                        grid.setPoint(i, j, k);
```

```
                                }
                            }
                        }
                    }

                    MainWindow win = new MainWindow();
                    win.setHelper(helper);
                    win.setVisible(true);

                }

            }
```

1.2 Panel

```
package application;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.ArrayList;

import javax.swing.JPanel;

import engine.IPaintable;

public class Panel extends JPanel{

	ArrayList<IPaintable> painters;

	public Panel() {
		setBackground(Color.WHITE);
		painters = new ArrayList<IPaintable>();
	}

	public void addPainter(IPaintable p) {
		painters.add(p);
	}

	public void paint(Graphics g2) {
		Graphics2D g = (Graphics2D) g2;

		g.setColor(Color.WHITE);
```

```
                g.fillRect(0, 0, 3000, 3000);

                g.setColor(Color.BLUE);

                for(int i = 0; i < painters.size(); i++) {
                        painters.get(i).paint(g);
                }
        }

}
```

1.3 MainWindow

```
package application;

import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.swing.*;

import engine.GridHelper;

public class MainWindow extends JFrame{
   private JMenuBar menu;
   private JMenu mBegin, mFile;
   private JMenuItem miStartOne, miStartMany, miStop,
miStartUpdates, miStopUpdates, miCalculateAtoms,
                                        miExportTxt, miExportVMD,
miBeginExport, miEndExport, miSave;
   private JLabel stepInfo, layerInfo;
   private Panel panel;

   public GridHelper helper;
   public boolean isUpdating;
// public ArrayList<IPaintable> paintedChildren;


   //Action section
```

```
    private ActionListener alStartOne = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
            helper.startCalculation();
        }
    };

    private ActionListener alStartMany = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
//          helper.startCalculation();
        }
    };

    private ActionListener alStop = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
            helper.stopCalculation();
        }
    };

    private ActionListener alStartUpdates = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
            startUpdates();
        }
    };

    private ActionListener alStopUpdates = new ActionListener() {
        @Override
```

```
            public void actionPerformed(ActionEvent arg0) {
                stopUpdates();
            }
    };

    private  ActionListener alCalculateAtoms = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {

            }
    };

    private ActionListener alExportTxt = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                try {

                } catch (Exception e) {
                    e.printStackTrace();
                }
//                helper.saveForUser("");
            }
    };

    private ActionListener alStartExportingVMD = new ActionListener()
{
            @Override
            public void actionPerformed(ActionEvent arg0) {
                try {
```

```
                } catch (Exception e) {
                        e.printStackTrace();
                }
                helper.beginExporting();
        }
    };

    private ActionListener alExportVMD = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
                try {
                        helper.exportForVMD();
                } catch (Exception e) {
                        e.printStackTrace();
                }
//              helper.saveForUser("");
        }
    };

    public MainWindow() {

        super();
        initialize();

        this.addKeyListener(new KeyListener() {

                @Override
                public void keyPressed(KeyEvent e) {
                        //JOptionPane.showMessageDialog(null,
e.getKeyCode());
```

```
                switch(e.getKeyCode()) {
                        case(38):{
                                if(helper.currentlyPaintedPlane <
helper.grid.dimZ - 1) {
                                        helper.currentlyPaintedPlane++;
                                }
                                break;
                        }
                        case(40):{
                                if(helper.currentlyPaintedPlane > 0) {
                                        helper.currentlyPaintedPlane--;
                                }
                                break;
                        }
                        case(107):{

  helper.setTemperature(helper.getTemperature() + 0.1);
                                break;
                        }
                        case(109):{

  helper.setTemperature(helper.getTemperature() - 0.1);
                                break;
                        }
                }
                setTitle( Integer.toString(
helper.currentlyPaintedPlane)  + "   " + helper.getTemperature() + "  " +
helper.grid.concentration);
                }
```

```
                @Override
                public void keyReleased(KeyEvent arg0) {
                        // TODO Auto-generated method stub

                }

                @Override
                public void keyTyped(KeyEvent e) {

                }

        });
        isUpdating = false;
}

public void setHelper(GridHelper Helper) {
        helper = Helper;
        panel.addPainter(Helper);
}

public void initialize() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(700, 500);
        setLocationRelativeTo(null);
        setExtendedState(MAXIMIZED_BOTH);
        setVisible(true);
        setLayout(null);

        panel = new Panel();
```

```
        panel.setBounds(0, 25, this.getBounds().width,
this.getBounds().height - 25);
        add(panel);

        menu = new JMenuBar();
        menu.setVisible(true);
        menu.setBounds(0, 0, this.getBounds().width, 25);
        //menu.setLayout(new FlowLayout());



        //mBegin section
        mBegin = new JMenu("Actions");
        //mBegin.setBounds(0, 0, 50, 20);
        menu.add(mBegin);

        miStartOne = new JMenuItem("Start one");
        miStartOne.addActionListener(alStartOne);
        mBegin.add(miStartOne);

        miStartMany = new JMenuItem("Start many");
        miStartMany.addActionListener(alStartMany);
        mBegin.add(miStartMany);

        miStop = new JMenuItem("Stop");
        miStop.addActionListener(alStop);
        mBegin.add(miStop);

        miStartUpdates = new JMenuItem("Start updates");
        miStartUpdates.addActionListener(alStartUpdates);
        mBegin.add(miStartUpdates);
```

```
        miStopUpdates = new JMenuItem("Stop updates");
        miStopUpdates.addActionListener(alStopUpdates);
        mBegin.add(miStopUpdates);

        miCalculateAtoms = new JMenuItem("Calculate atoms");
        miCalculateAtoms.addActionListener(alCalculateAtoms);
        mBegin.add(miCalculateAtoms);

        //end of mBegin section


        //mFile section

        mFile = new JMenu("File");
        //mBegin.setBounds(0, 0, 50, 20);
        menu.add(mFile);

        miExportTxt = new JMenuItem("Export to .txt file");
        miExportTxt.addActionListener(alExportTxt);
        mFile.add(miExportTxt);

        miExportVMD = new JMenuItem("Export for VMD");
        miExportVMD.addActionListener(alExportVMD);
        mFile.add(miExportVMD);

        miBeginExport =  new JMenuItem("Start Exporting");
//      miBeginExport.addActionListener(alStartVMDAutosaves);
        mFile.add(miBeginExport);
```

```
        miEndExport =  new JMenuItem("Stop Exporting");
//      miEndExport.addActionListener(alStopVMDAutosaves);
        mFile.add(miEndExport);

        miSave = new JMenuItem("Save as backup");
//      miSave.addActionListener(alSaveGrid);
        mFile.add(miSave);

        menu.add(mFile);



        add(menu);
        menu.updateUI();
    }

    public void startUpdates() {
        isUpdating = true;
        Thread t = new Thread() {
            public void run() {
                while(isUpdating) {
                    panel.repaint();
                    try {
                        Thread.currentThread().join(1);
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }
```

```
            };
            t.start();
    }
    public void stopUpdates() {
            isUpdating = false;
    }
}
```

2 Пакет ядра

2.1 Інтерфейс відмальовки:

```
package engine;

import java.awt.Graphics2D;

public interface IPaintable {
   public void paint(Graphics2D g);
}
```

2.2 Структури даних про положення в просторі:

```
package engine;

public class PositionData {
   public int x, y, z;
   public double weight;

   public PositionData() {
         setDefaultValue();
   }

   public PositionData(int x, int y, int z) {
         this.x = x;
         this.y = y;
         this.z = z;
         weight = 0;
   }

   public PositionData(int x, int y, int z, double weight) {
         this.x = x;
```

```
        this.y = y;
        this.z = z;
        this.weight = weight;
    }

    public void setDefaultValue() {
        x = 0;
        y = 0;
        z = 0;
        weight = 0;
    }

    public void setValue(int x, int y, int z, double weight) {
        this.x = x;
        this.y = y;
        this.z = z;
        this.weight = weight;
    }
}
```

3 Ядро:

```
package engine;

import java.util.ArrayList;
import java.util.Random;

import Helpers.Point;

public class Grid {

    public byte[][][] grid;
    public ArrayList<Point> queue;
    public Random random;
    public int dimX, dimY, dimZ;
    public double concentration;
    //#region environment parameters


    public double alpha = 2;
    public double p0 = 0.685;

    //#endregion


    private boolean[][][] validationPattern;


    //#region optimization garbage section

    private int neirbourghsCountResult;
```

```
    private PositionData collectedPositionData;


    //#endregion optimization garbage section

    public Grid() {
        this(100, 100, 100);
    }

    public Grid(int DimX, int DimY, int DimZ) {
        dimX = DimX;
        dimY = DimY;
        dimZ = DimZ;

        validationPattern = new boolean[5][5][5];

        for(int i = 0; i < validationPattern.length; i++) {
            for(int j = 0; j < validationPattern[0].length; j++) {
                for(int k = 0; k < validationPattern[0][0].length;
k++) {
                    validationPattern[i][j][k] = false;
                }
            }
        }

        validationPattern[0][0][0] = true;
        validationPattern[0][0][4] = true;
        validationPattern[0][4][0] = true;
        validationPattern[4][0][0] = true;
        validationPattern[0][4][4] = true;
```

```
        validationPattern[4][0][4] = true;
        validationPattern[4][4][0] = true;
        validationPattern[4][4][4] = true;

        validationPattern[2][2][0] = true;

        validationPattern[1][1][1] = true;
        validationPattern[3][3][1] = true;

        validationPattern[2][0][2] = true;
        validationPattern[0][2][2] = true;
        validationPattern[4][2][2] = true;
        validationPattern[2][4][2] = true;

        validationPattern[1][3][3] = true;
        validationPattern[3][1][3] = true;

        validationPattern[2][2][4] = true;


        grid = new byte[dimX][dimY][dimZ];
        queue = new ArrayList<Point>();
        random = new Random();

        concentration = 0.001077 / 4;


    }

    public void setAlpha(double value) {
```

```
        alpha = value;
        double ep0 = Math.pow(Math.E, alpha);
        p0 = Math.pow(0.685, alpha); // 0.685

        System.out.println(String .format("alpha: %s; p0: %s", alpha,
p0));

    }

    public double getAlpha() {
        return alpha;
    }

    //region

    //marks dimensions and grid specifics
    public boolean isValid(int x, int y, int z) {
        return(
                (x >= 0 && y >= 0 && z >= 0 && x < dimX && y
< dimY && z < dimZ)
                && validationPattern[x % 4][y % 4][z % 4]
                //&& (x - dimX*0.5)*(x - dimX*0.5) + (y -
dimY*0.5)*(y - dimY*0.5) < dimX*dimX * 0.25
            );
    }

    public void setPoint(int i, int j, int k) {
        if(isValid(i, j, k)) {
            grid[i][j][k] = 10;
            queue.add(new Point(i, j, k));
```

```
		}
	}

	public void setPointUnbound(int i, int j, int k) {
		if(isValid(i, j, k)) {
			grid[i][j][k] = 1;
			queue.add(new Point(i, j, k));
		}
	}

	public void setPointUnchecked(int i, int j, int k) {
		if(isValid(i, j, k)) {
			grid[i][j][k] = 127;
			//queue.add(new Point(i, j, k));
		}
	}

	public int getNeirbourghsCount(int x, int y, int z) {
//		int count = 0;
		neirbourghsCountResult = 0;

		for(int i = -1; i <= 1; i++ ) {
			for(int j = -1; j <= 1; j++ ) {
				for(int  k = -1; k <= 1; k++) {
					if(isValid(x + i, y + j, z + k)
							&& (i != 0 && j != 0 && k !=
0)
							&& grid[x + i][y + j][z + k] > 9)
{
//						count++;
```

```
                                        neirbourghsCountResult++;
                                    }
                                }
                            }
                        }
//                      return count;
                        return neirbourghsCountResult;
    }

    public boolean hasBoundedNeirbourghs(int x, int y, int z) {
                        for(int i = -1; i <= 1; i++ ) {
                            for(int j = -1; j <= 1; j++ ) {
                                for(int  k = -1; k <= 1; k++) {
                                    if(isValid(x + i, y + j, z + k)
                                            && (i != 0 && j != 0 && k !=
0)
                                            && grid[x + i][y + j][z + k] > 9)
{
                                        return true;
                                    }
                                }
                            }
                        }
                        return false;
    }

    /*public int getPositionWeight(int x, int y, int z) {

                        int count = 0;
```

```
        for(int i = -1; i <= 1; i++ ) {
            for(int j = -1; j <= 1; j++ ) {
                for(int  k = -1; k <= 1; k++) {
                    if(isValid(x + i, y + j, z + k)
                            && (i != 0 && j != 0 && k !=
0)
                            && grid[x + i][y + j][z + k] > 9
                            && getNeirbourghsCount(x + i,
y + j, z + k) > 0) {
                        count++;
                    }
                }
            }
        }
        return count;
    }*/

    public ArrayList<PositionData> getBoundedPositionData(int x, int y,
int z){
        ArrayList<PositionData> data = new
ArrayList<PositionData>();

        for(int i = -1; i <= 1; i++ ) {
            for(int j = -1; j <= 1; j++ ) {
                for(int  k = -1; k <= 1; k++) {

                    if(isValid(x + i, y + j, z + k)
                            && !(i == 0 && j == 0 && k
== 0)
```

```
                                        && grid[x + i][y + j][z + k] ==
0
                                        ) {
                                    data.add(new PositionData(x + i, y + j,
z + k,
                                            getNeirbourghsCount(x +
i, y + j, z + k) != 0 ? Math.exp( alpha * (getNeirbourghsCount(x + i, y +
j, z + k))) : 0
                                            ));
                                }
                            }
                        }
                    }

                    data.add(new PositionData(x, y, z, Math.exp( alpha *
(getNeirbourghsCount(x, y, z)))));

                    return data;
                }

                public ArrayList<PositionData> getFreePositionData(int x, int y, int
z){
                    ArrayList<PositionData> data = new
ArrayList<PositionData>();

                    for(int i = -1; i <= 1; i++ ) {
                        for(int j = -1; j <= 1; j++ ) {
                            for(int  k = -1; k <= 1; k++) {

                                if(isValid(x + i, y + j, z + k)
```

```
                                        && grid[x + i][y + j][z + k] ==
0
                                        ) {

                                    data.add(new PositionData(x + i, y + j,
z + k,
                                            1
                                            ));

                                }
                            }
                        }
                }

                data.add(new PositionData(x, y, z, 1));

                return data;
        }

        public PositionData getNextPosition(int x, int y, int z) {
                ArrayList<PositionData> data;
//              if( grid[x][y][z] < 10 ) {
//                      data = getFreePositionData(x, y, z);
//              }
//              else {
//                      data = getBoundedPositionData(x, y, z);
//              }

                //if( grid[x][y][z] > 9 || hasBoundedNeirbourghs(x, y, z))
                {
```

```
                data = getBoundedPositionData(x, y, z);

        }
//      else
//      {
//              data = getFreePositionData(x, y, z);
//      }



        double amplitude = 0;

        for(int i = 0; i < data.size(); i++) {
                amplitude += data.get(i).weight;
        }

        double point = amplitude * random.nextDouble();
        double sum = 0;

        for(int i = 0; i < data.size(); i++) {
                sum += data.get(i).weight;
                if(sum >= point) {
                        return data.get(i);
                }
        }

        return data.get(data.size() - 1);
    }

    //from, where
```

```
public void move(Point p, int i, int j, int k) {

        grid[p.x][p.y][p.z] = 0;
        boolean globalHandle = hasBoundedNeirbourghs(i, j, k);
        if(!globalHandle || isTopRegion(i, j, k)) {
                grid[i][j][k] = 1;
        }
        else {
                grid[i][j][k] = 10;
        }

        p.x = i;
        p.y = j;
        p.z = k;



        /*boolean globalHandle = hasBoundedNeirbourghs(i, j, k);
        if(!globalHandle || isTopRegion(i, j, k)) {
                grid[i][j][k] = 1;
        }
        else {
                grid[i][j][k] = 10;
        }
        grid[p.x][p.y][p.z] = 0;
        p.x = i;
        p.y = j;
        p.z = k;*/
}

public void jump(Point p) {
```

```
            int count = getNeirbourghsCount(p.x, p.y, p.z);
            if( count < 4 && random.nextDouble() < Math.pow(p0,
count)/*Math.exp(-1 * p0 * count) */) {
                  PositionData data = getNextPosition(p.x, p.y, p.z);
                  if(grid[data.x][data.y][data.z] == 0) {
                        move(p, data.x, data.y, data.z);
                  }

            }
      }

      public void rearrange() {

            int x, y;
            Point temp;
            for(int i = 0; i < queue.size() / 2 ; i++) {
                  x = random.nextInt(queue.size());
                  y = random.nextInt(queue.size());

                  temp = queue.get(x);
                  queue.set(x, queue.get(y));
                  queue.set(y, temp);

                  temp = null;
            }
      }

      //endregion

      public boolean isTopRegion(int i, int j, int k) {
```

```
        return k > dimZ* 0.75;
    }

    public void refillTopRegion(double probability) {
        for(int i = 0; i < dimX; i++) {
            for(int j = 0; j < dimY; j++) {
                for(int k = 0; k < dimZ; k++) {
                    if(isValid(i, j, k) && isTopRegion(i, j, k)
&& grid[i][j][k] == 0 && random.nextDouble() < probability) {
                        setPoint(i, j, k);
                    }
                }
            }
        }
    }

    public double probab() {
        double count = 0;
        double volume = 0;//(int) (dimX * dimY * dimZ * 0.2);


        for(int i = 0; i < dimX; i++) {
            for(int j = 0; j < dimY; j++) {
                for(int k = 0; k < dimZ; k++) {
                    if(isTopRegion(i, j, k)) {
                        if(grid[i][j][k] != 0) {
                            count++;
                        }
                        if(isValid(i, j, k)) {
```

```
                                        volume++;
                                    }
                                }
                            }
                        }
                    }

                //System.out.println(count + " / " + volume + " count/volume");
                //0.015508
                return concentration - count / volume;
//;//0.00128 * 4.0 - count / volume;//0.00017657 - count /
volume;//0.00009625 - count / volume;
    }

}
```

2.4 Обгортка, що працює із ядром:

```
package engine;

import java.awt.Color;
import java.awt.Graphics2D;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;

import Helpers.Point;
import logging.Logger;
import logging.Writable;
```

```
public class GridHelper implements IPaintable{

   public Grid grid;
   public boolean calculationRunning;
   public boolean snapshotsCreating;

   //#region paint settings

   public int currentlyPaintedPlane, halfSize = 2;

   //#endregion

   public GridHelper() {
         calculationRunning = false;
         currentlyPaintedPlane = 0;
   }


   public void exportForVMD() throws IOException {
         Writable snapshot = new Writable() {

               private String name = "fsds.pdb";

               @Override
               public void write(BufferedWriter writer) throws
Exception {
                     try {
                           exportForVMD(writer);
                     }
                     catch(Exception ex) {
```

```
                                Logger.log.println(String.format("%s",
ex.getStackTrace()));
                        }
                }

                @Override
                public String getName() {
                        return name;
                }

                @Override
                public void setName(String name) {
                        this.name = name;
                }

        };

        Logger.log.logDebugSnapshot(snapshot);
  }

  public void exportForVMD(BufferedWriter writer) throws
IOException {
        synchronized(grid.grid) {
                for(Point p : grid.queue) {

                        if(/*grid.getNeirbourghsCount(p.x, p.y, p.z) >
9)*/grid.grid[p.x][p.y] [p.z] > 9)
                        {
```

```
                                writer.write("ATOM    100  N   VAL A  25
" + (form(10*p.x)) + " " + (form(10*p.y)) + " " + (form(10*p.z)) + "
1.00 12.00      A1   C   " + System.lineSeparator());
                        }
                        else
                        {
                                //writer.write("ATOM    100  B   VAL A  25
" + (form(10*p.x)) + " " + (form(10*p.y)) + " " + (form(10*p.z)) + "
1.00 12.00      A1   C   " + System.lineSeparator());
                        }
                }

        }
  }

  public String form(int number) {
        String s = Double.toString(number / 10.0);

        while (s.indexOf(".") < 3) {
                s = "0" + s;
        }

        while (s.length() < 7) {
                s += "0";
        }

        return s;
  }

  public void startCalculation() {
```

```
calculationRunning = true;
Thread t = new Thread() {
        public void run() {
                grid.rearrange();

                double size = grid.queue.size();
                double step = 0;

                Writable snapshot = new Writable() {

                        private String name;

                        @Override
                        public void write(BufferedWriter writer)
throws Exception {
                                try {
                                        exportForVMD(writer);
                                }
                                catch(Exception ex) {

  Logger.log.println(String.format("%s", ex.getStackTrace()));
                                }
                        }

                        @Override
                        public String getName() {
                                return name;
                        }

                        @Override
```

```
                    public void setName(String name) {
                        this.name = name;
                    }

                };

                while(calculationRunning) {

                    Logger.log.println(String.format("Step: %s",
step));
                    snapshot.setName(String.format("%s",
step));
                    Logger.log.logSnapshot(snapshot);

                    for(int k = 0; k < 10000; k++)
                    {
                        for(int i = 0; i < grid.queue.size(); i++)
{
                            synchronized(grid.grid) {

  grid.jump(grid.queue.get((grid.random.nextInt(grid.queue.size()))));
                            }
                        }

                    }

                    step += 10000;

                }
            }
```

```
        };
        t.start();
}

public void stopCalculation(){
        calculationRunning = false;
}

public void beginExporting() {
        snapshotsCreating = true;
}

public void endExporting() {
        snapshotsCreating = false;
}

public void rearrangeGridQueue() {
        if(grid != null) {
                int x, y; //future coordinates of elements
                Point temp;

                for(int i = 0; i < grid.queue.size(); i++) {
                        x = grid.random.nextInt(grid.queue.size());
                        y = grid.random.nextInt(grid.queue.size());

                        temp = grid.queue.get(x);
                        grid.queue.set(x, grid.queue.get(y));
                        grid.queue.set(x, temp);
                }
        }
```

```
	}

	public double getTemperature() {
		return grid.alpha;
	}

	public void setTemperature(double d) {
		grid.setAlpha(d);
	}

	@Override
	public void paint(Graphics2D g) {
		g.setColor(Color.WHITE);
		g.fillRect(0, 0, 10000, 10000);

		g.setColor(Color.BLUE);

		for(int i = 0; i < grid.dimX; i++) {
			for(int j = 0; j < grid.dimY; j++) {

				if(grid.grid[i][j][currentlyPaintedPlane] != 0) {
					g.fillRect( ( i) * halfSize, (  j) * halfSize, 2 *
halfSize, 2 * halfSize);
				}
				else {
//					g.setColor(Color.GRAY);
//					g.fillRect( ( 2 *i) * halfSize + 1, ( 2* j) *
halfSize + 1, 2 * halfSize - 2, 2 * halfSize - 2);
				}
```

```
                    }
                }

            }

        }
```

3 Допоміжні математичні структури ядра:

```
package Helpers;

public class Point{

	public int x, y, z;

	public Point() {
		setDefaultValue();
	}

	public Point(int x, int y, int z) {
		this.x = x;
		this.y = y;
		this.z = z;
	}

	public void toDPoint(DPoint p) {
		p.x = x;
		p.y = y;
		p.z = z;
	}

	public void setDefaultValue() {
		x = 0;
		y = 0;
		z = 0;
	}

}
```

```
package Helpers;

public class DPoint {
	public double x;
	public double y;
	public double z;

	public DPoint() {
		setDefaultValue();
	}

	public DPoint(double x, double y, double z) {
		this.x = x;
		this.y = y;
		this.z = z;
	}

	public void toPoint(Point p) {
		p.x = (int) x;
		p.y = (int) y;
		p.z = (int) z;
	}

	public void setDefaultValue() {
		x = 0;
		y = 0;
		z = 0;
	}
}
```

4 Засоби збереження даних:

4.1 Інтерфейс для взаємодії:

```
package logging;

import java.io.BufferedWriter;

public interface Writable {
	public void write(BufferedWriter writer) throws Exception;
	public String getName();
	public void setName(String name);
}
```

4.2 Засіб для запису:

```
package logging;

import java.io.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Logger {
	public static final Logger log = new Logger();



	private static final String ROOT_FOLDER = "Logs";
    private final String ExperimentDataContainer =
String.format("Experiment_[%s]", getTimestamp());
    private final String snapshotFolderName = "Snapshots";
    private final String snapshotInvariantNamePart = "Snapshot_";
    private final String LogFileName = "Log.txt";
```

```
    private BufferedWriter integrityLogWriter;
    private BufferedWriter temperatureLogWriter;
    private BufferedWriter logWriter;


    protected Logger()
    {
        try {
                    initializeLoggingSession();
                } catch (IOException e) {
                    e.printStackTrace();
                }
    }

    private void initializeLoggingSession() throws IOException {
        File snapshotsFolder = new File(String.format("%s\\%s\\%s\\",
ROOT_FOLDER, ExperimentDataContainer, snapshotFolderName));
        snapshotsFolder.mkdirs();

        logWriter = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s", ROOT_FOLDER, ExperimentDataContainer,
LogFileName)), true));
    }

    public void println(String s) {
        String message = String.format("Time: %s <nanos: %s>|  %s",
getTimestamp(), System.nanoTime(),  s);
        System.out.println(message);
        try {
```

```
                logWriter.write(message);
                logWriter.newLine();
                logWriter.flush();
            } catch (Exception e) {
                System.err.println(String.format("Failed to append stream:
<logWriter> \r\n    %s", (Object)e.getStackTrace() ));
            }
    }

    public static String getTimestamp() {
        SimpleDateFormat sdfDate = new SimpleDateFormat("yyyy MM dd
(HH_mm_ss)");
            Date now = new Date();
            return sdfDate.format(now);
    }

    public void logSnapshot(Writable loggedOject) {
        Logger.log.println("Logging snapshot...");
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(new
File(String.format("%s\\%s\\%s\\%s%s",
                                ROOT_FOLDER,
                                ExperimentDataContainer,
                                snapshotFolderName,
                                snapshotInvariantNamePart,
                                loggedOject.getName())), true));

            loggedOject.write(writer);
            writer.flush();
            writer.close();
```

```
        }
        catch(Exception e) {
                System.err.println(e.getMessage());
        }
        Logger.log.println("Logging snapshot finished");
    }

    public void logDebugSnapshot(Writable loggedOject) {
        Logger.log.println("Logging debug snapshot...");
        try {
                BufferedWriter writer = new BufferedWriter(new FileWriter(new
File(loggedOject.getName()), true));

                loggedOject.write(writer);

                        writer.flush();
                writer.close();
        } catch (Exception e) {
                        e.printStackTrace();
                }
        finally {
                Logger.log.println("Debug snapshot is logged");
        }
    }

    @Override
    public void finalize() {

        try {
                integrityLogWriter.flush();
```

```
            integrityLogWriter.close();
        }
        catch(Exception e) {

        }

            try {
                temperatureLogWriter.flush();
                temperatureLogWriter.close();
        }
        catch(Exception e) {

        }

            try {
                logWriter.flush();
                logWriter.close();
            }
            catch(Exception e) {

            }

    }

}
```